Sorting Algorithms

Peter Tiño

P.Tino@cs.bham.ac.uk, www.cs.bham.ac.uk/~pxt

School of Computer Science
University of Birmingham, UK
Assume an ordered set $A$ - e.g. there is a natural (linear) order defined on the elements of the set.

Examples:
- \{A, B, C, \ldots, Z\}, $A < B < C \ldots < Z$
- $\mathbb{R}$, usual order $x < y$

We are given a (possibly long) list (sequence) $S$ of elements from $A$.

Need to permute $S$ so that the elements are in non-decreasing (non-increasing) order.

Important for a variety of algorithms operating on $S$.

We will go through some of the suggestions of how to do this (efficiently).
Let’s start simple... Insertion Sort

In each iteration:
- remove one element from the list
- find its location within the sorted list so far
- insert it there
- repeat until no input elements remain.

sorted so far, shifted by 1 position →

7, 4, 2, 8, 3, 1, 5, 6
7, x, 2, 8, 3, 1, 5, 6
4, 7, 2, 8, 3, 1, 5, 6
4, 7, x, 8, 3, 1, 5, 6
2, 4, 7, 8, 3, 1, 5, 6
2, 4, 7, x, 3, 1, 5, 6
2, 4, 7, 8, 3, 1, 5, 6
2, 4, 7, 8, x, 1, 5, 6
2, 3, 4, 7, 8, 1, 5, 6

etc.
Insertion Sort

Abstract representation:

\[ \leq a, > a, \ a, \text{ unsorted elements} \]

\[ \leq a, > a, x, \text{ unsorted elements} \]

\[ \leq a, a, > a, \text{ unsorted elements} \]

Think of the best/worst cases (number of steps needed).
**Insertion Sort**

**Best case:** already sorted - 1, 2, 3, 4, 5, 6, 7, 8  
Why?  
No shift-and-insert actions needed. The first element from unsorted items is compared only with the right-most element of the sorted items.  

**Worst case:** reverse order - 8, 7, 6, 5, 4, 3, 2, 1  
Why?  
Many shift-and-insert actions are needed. The first element from unsorted items is always smaller than sorted items.
Big O notation describes limiting behavior of a function ("order" of the function).

It characterises functions with respect to their "growth rates". The idea is that different functions with the same growth rate get represented by the same O notation.

What does it mean to write $f(n) \in O(g(n))$?
There is a constant $c > 0$ and some $n_*$, after which $f(n) < c \cdot g(n)$. In other words, for all $n > n_*$, we have $f(n) < c \cdot g(n)$.

Can you determine O notation for the number of steps Insertion Sort needs in the best/worst cases (lists of size $n$).
**Best case:** already sorted - 1,2,3,4,5,6,7,8
What is the number of steps required? \([O(n)]\)
No shift-and-insert actions needed. The first element from unsorted items is compared only with the right-most element of the sorted items.

**Worst case:** reverse order - 8,7,6,5,4,3,2,1
What is the number of steps required? \([O(n^2)]\)
Many shift-and-insert actions are needed. The first element from unsorted items is always smaller then sorted items.

\(N\) comparisons + \(1 + 2 + 3 + \ldots + (n - 1)\) shifts
\(1 + 2 + 3 + \ldots + n = n(n + 1)/2\) operations
Selection Sort

sorted so far, unsorted, min element of unsorted

In each iteration:
- find min element from unsorted list
- swap it with the left-most unsorted element
- repeat until unsorted list is empty.

7, 4, 2, 8, 3, 1, 5, 6
7, 4, 2, 8, 3, 1, 5, 6
1, 4, 2, 8, 3, 7, 5, 6       swapped 7 with 1
1, 4, 2, 8, 3, 7, 5, 6
1, 4, 2, 8, 3, 7, 5, 6
1, 2, 4, 8, 3, 7, 5, 6       swapped 4 with 2
1, 2, 4, 8, 3, 7, 5, 6
1, 2, 4, 8, 3, 7, 5, 6
1, 2, 3, 8, 4, 7, 5, 6       swapped 4 with 3
1, 2, 3, 8, 4, 7, 5, 6
1, 2, 3, 8, 4, 7, 5, 6
etc.
**Bubble Sort**

*In each iteration:*
- go through the list to be sorted
- for each adjacent couple - swap if not in the right order
- repeat passes through the list until no more swaps are needed.

**couple to check, swapped couple**

7, 4, 2, 8, 3, 1, 5, 6
7, 4, 2, 8, 3, 1, 5, 6
4, 7, 2, 8, 3, 1, 5, 6
4, 7, 2, 8, 3, 1, 5, 6
4, 2, 7, 8, 3, 1, 5, 6
4, 2, 7, 8, 3, 1, 5, 6
4, 2, 7, 8, 3, 1, 5, 6
no swap needed
4, 2, 7, 8, 3, 1, 5, 6
4, 2, 7, 3, 8, 1, 5, 6
etc.
Bubble Sort

After 1st pass: 4, 2, 7, 3, 1, 5, 6, 8

Second pass:

4, 2, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8
2, 4, 7, 3, 1, 5, 6, 8

go for the 3rd pass, etc.
**Comb Sort**

**Bubble Sort** has a problem - if there are many small values towards the end of list (turtles) - sorting is considerably slowed down. Why?

The problem is that at each stage of sorting we can only swap two adjacent elements (if they are not in the right order). It can take a long time for a small element to propagate to the beginning of the list.

**Comb Sort** allows the distance between the two compared elements be greater than 1 (unlike in Bubble Sort, where it is always 1).

The gap between swapped elements decreases (after each pass through the list) by the shrink factor $r$. Typically, $r = 1.3$.

For the $i$-th pass through the list the gap size is $n \cdot r^{-i}$ (appropriately rounded, until it hits 1; after that it stays 1 (Bubble sort)).